Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Methods and Techniques for Discovering Taxonomies of Behavioral Process Models

Francesco Folino[1], Gianluigi Greco[2],
Antonella Guzzo[3], Luigi Pontieri[1]

RT-ICAR-CS-09-08 **Settembre 2009**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Methods and Techniques for Discovering Taxonomies of Behavioral Process Models

Francesco Folino[1], Gianluigi Greco[2],
Antonella Guzzo[3], Luigi Pontieri[1]

Rapporto Tecnico N.: **RT-ICAR-CS-09-08**

Data: **Settembre 2009**

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Cosenza, Via P. Bucci 41C, 87036 Rende(CS)
[2] Università degli Studi della Calabria, Dipartimento di Matematica, Via P. Bucci 30B, Rende (CS)
[3] Università degli Studi della Calabria, Dipartimento di Elettronica, Informatica e Sistemistica, Via P. Bucci 41C, Rende (CS)

Methods and Techniques for Discovering Taxonomies of Behavioral Process Models

Francesco Folino, ffolino@icar.cnr.it ICAR-CNR
Gianluigi Greco, ggreco@mat.unical.it University of Calabria
Antonella Guzzo, guzzo@deis.unical.it University of Calabria
Luigi Pontieri, pontieri@icar.cnr.it ICAR-CNR

Keywords

Taxonomy of Data, Process Mining, Abstraction, Knowledge Discovery, Workflows

Abstract

Modelling behavioral aspects of business processes is a hard and costly task, which usually requires heavy intervention of business experts. This explain the increasing attention given to process mining techniques, which automatically extract behavioral process models from log data. In the case of complex processes, however, the models identified by classical process mining techniques are hardly useful to analyze business operations at higher abstraction levels. In fact, the need of process abstraction emerged in several application scenarios, and abstraction methods are already supported in some business-management platforms, which allow users to manually define abstract views for the process at hand. Therefore, it comes with no surprise that process mining research has recently considered the issue of mining processes at different abstraction levels, mainly in the form of a taxonomy of process models, as to overcome the drawbacks of traditional approaches. This paper offers a survey on these recently proposed mechanisms, including: (i) workflow modelling and discovery techniques, (ii) clustering techniques enabling the discovery of different behavioral process classes, and (iii) activity abstraction techniques for associating a generalized process model with each higher level taxonomy node.

Introduction

Workflow models are an effective way to specify the behavior of complex processes in terms of elementary process activities and routing constructs (e.g. parallelism, loops, splits). Unfortunately, modelling the behavioral aspects of a business process is a time-consuming task, requiring heavy intervention by business experts. This motivates the interest for process mining techniques (see Reference 41 for a survey on this topic), which allow to automatically extract a workflow model based on the execution logs available for a given process. Traditional process mining approaches are designed to support process enactments, and thus define workflow models that completely specify all the operational details for the process. In practice, however, business users want to analyze business operations at higher abstraction levels. In fact, the need of process abstraction emerged in several applications, and it is supported by some business-management platforms (e.g, iBOM[8], ARIS[25]), which allow the user to manually define abstract views for the process.

A first step towards the automatic construction of process model taxonomies via induction methods was done in Reference 14, where different behavioral classes of process instances are discovered with a clustering method, and are eventually equipped with separate workflow models. Indeed, such a result can be used as a basis for restructuring the representation of the process behavior into a taxonomy of process models, by way of suitable abstraction techniques[13]. Other approaches have also subsequently been proposed, so that a number of methods and algorithms currently exist, which are support process mining from the perspective of deriving abstract views for the processes at hand.

This work is precisely intended to provide a survey on different kinds of techniques and methods that can support the (automatic) design of process model taxonomies effectively. The following sections are organized as follows. In the first section, a few preliminaries on process models and process mining are discussed. The second section introduces the concept of process model taxonomy, and refers to diverse notions of specialization proposed in the literature for behavioral process models. The usage of clustering techniques is investigated in the third section, as a basic means for recognizing different behavioral classes. Eventually, the fourth section illustrates some abstraction mechanisms that allow to transform a process model into a more general one, which can be exploited to derive an abstract model for each higher level taxonomy nodes.

Workflow-based Representation and Mining of Process Behavior

Workflow models (more precisely control-flow models) are quite a popular and intuitive way of representing process behavior, where legal ways of executing process activities are encoded in terms of precedence constraints and more elaborate routing constructs (such as concurrence, loops, synchronization and choice). In this section, we introduce some basic concepts on workflow models, with the help of a simplified modelling language. Then, we discuss some major mining techniques which can induce such a model automatically from execution logs, and can hence effectively support the design of behavioral process models.

Workflow Models and Logs

Workflow model represents all possible execution flows along the activities of a given process, by means of a set of constraints defining “legal" execution in terms of simple relationships of precedence and/or more elaborate constructs such as loops, parallelism, synchronization and choice (just to cite a few). A significant amount of research has been done in the context of specification mechanisms for process modelling (e.g., *Event Driven Process Chains*[40], *Petri Nets*[39], and others[12,35]). In particular, Petri nets were largely used in workflow management applications for they can various routing constructs, yet enjoying precise execution semantics.

In order to keep the exposition clear and concrete, hereinafter we will refer to a simplified and intuitive modelling language for workflow models, where precedence relationships between activities are depicted as arrows between nodes of a *workflow* graph, and where some constructs drawn by means of labels beside the tasks (cf. *AND*, *OR*, *XOR*) are used to state more elaborate constraints of execution. Roughly speaking, a node whose input is *AND* acts as synchronizer (i.e. it can be executed only after all its predecessors have been completed), whereas a node whose input is *OR* can start as soon as at least one of its predecessors has been completed. Furthermore, once finished, a node with an *AND* (resp., *OR*, *XOR*) in output activates all (resp., some of, exactly one of) its outgoing activities.

Example As an example scenario, let us consider the process of handling customers’ orders within a business company. Fig. 1 shows a workflow model for the above process, where edges represent precedence relationships, while additional constraints are expressed via labels associated with activity nodes. E.g., task l is an AND-join activity, as it must be notified that both the client is reliable and the order can be supplied correctly. Conversely, b is a XOR-split activity, since it can activate just one of its adjacent activities.

Figure 1

Workflow model for the sample *HandleOrder* process

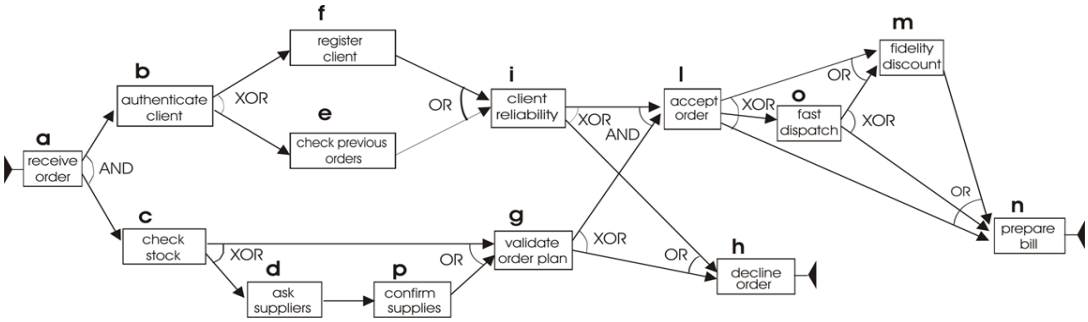


Figure 2

Sample log for the *HandleOrder* process.

s_1 : acbgfh	s_9 : abefcgil
s_2 : abfcgh	s_{10} : acgbefijl
s_3 : acgbfh	s_{11} : abcedfgil
s_4 : abcgfil	s_{12} : acdbefgil
s_5 : abfcgikl	s_{13} : abcfdgikl
s_6 : acbfgijl	s_{14} : acdbfgikl
s_7 : acbgfijkl	s_{15} : abcdgfikl
s_8 : abcegfijl	s_{16} : acbfdgil

It is worth noting that most of the methods discussed in this paper are orthogonal to the language adopted to represent process behavior, and do not depend on the simplistic notation introduced above. Hence, for generality we may assume that a set of activity identifiers, say *A*, is given and that a modelling language, say *M*, is used. Then, we shall denote by *M*(*A*) the set of *workflow models* that can be built by using the constructs in *M* over the activities in *A*.

Each time a workflow model $W \in M(A)$ is enacted in a workflow management system, the activities in *A* are executed according to the constraints of *W*, till some final configuration is reached for which there is no constraint in *W* forcing the execution of some further activity. The events related to each enactment may be stored in some log repository, and used as input for *process mining* techniques, which allow to induce a workflow model $W \in M(A)$ even in the case the original workflow model *W* is not known or it does not exists at all (i.e., the process is carried out without any formal and explicit execution model). In fact, most process-oriented systems store information on process instances, by keeping track of the activities executed during each of them. Basically, a process log can be seen as a set of *traces*, which, in the most simplistic scenario, correspond to strings over activity identifiers, representing ordered sequences of activities. A log of few traces is shown in Fig. 2, for the example process *HandleOrder*.

Workflow Discovery Techniques

Based on execution logs like the ones introduced before, process mining techniques (see Reference 41 for a survey) allow to discover knowledge about the behavior of the processes that generated them. We here concentrate process mining techniques (known as control-flow mining techniques) that aim at discovering a workflow representation for the behavior registered in a given execution log. Most of the differences among the different proposals in the literature resides in the modelling features that can be used to represent a workflow model and in the specific algorithms used for discovering it. For example, in Reference 2, processes are intuitively represented through pure directed graphs, which allow to express precedence relationships only, while disregarding richer control flow constructs, such as concurrency, synchronization and choice. Many other proposals exploit, instead, more expressive

languages, which sometimes enjoy deep formal foundation for modelling and analyzing workflow processes, as in the case of WF-nets[37,41,42] (a special kind of Petri nets).

The general problem of discovering a WF-net workflow model was specifically analyzed in Reference 42, where the concept of *structured workflow* (*SWF*) net is introduced to capture a class of WF-nets that a process mining algorithm should be able to rediscover. Here an algorithm, named α is presented which can rediscover an *SWF* net out of a log, provided that the log is guaranteed to enjoy some well-specified properties. The α algorithm was then extended[10] with some pre-processing and post-processing strategies making it capable to discover short loops. Moreover another extension[38] of the algorithm was devised to explicitly capture non-free-choice constructs, which are a form of implicit dependencies between the process tasks.

A heuristic approach[37] was presented that exploits simple metrics concerning task dependency and task frequency, in order to produce a graph-based model, called “dependency/frequency graph". Notably, the approach copes with the presence of noisy logs.

A different approach to mining a process model from event logs was described in Reference 34, in order to discover a hierarchically structured workflow model. Such a model corresponds to an expression tree, where the leafs represent tasks (operands) while any other node is associated with a control flow operator. In this context, the mining algorithm mainly consists of suitable term rewriting systems.

Yet another approach was adopted in References 23,24, where a subset of the ADONIS[26] definition language is used to represent a block-structured workflow model. The peculiarity of the approach mainly resides in its capability of recognizing duplicate tasks in the control flow graph, i.e., many nodes associated with the same task. The algorithm proposed there, named “InWoLvE", solves the process mining problem in two steps: an induction step, where a stochastic activity graph (*SAG*) is extracted out of the input log, and a transformation step, where the *SAG* is transformed into a block-structured workflow-model.

Recently, some extensions to this approach were proposed in Reference 21, in an interactive setting, where the analyst can iteratively refine the results found by evaluating the mined models and varying the parameters of the mining technique. Beside discussing a number of issues involved in interactive process mining, some techniques are introduced in Reference 21 to support such a setting, which primarily include a validation procedure for checking the (preliminary) mined model, and a structured layout algorithm that is stable against small changes of the mined model.

The problem of discovering a process model from execution logs was also considered in Reference 7, as a special case of the *Maximal Overlap Sets* problem in graph matching. The paradigm of planning and scheduling by resource management is used there in order to devise an efficient approach tackling the combinatorial complexity of the problem.

The approach proposed in Reference 11 tries to overcome the difficulty encountered by previous techniques when dealing with non-trivial routing constructs and noisy data, by resorting to the use of genetic algorithms. Indeed, this kind of algorithm offers a way to discover non-trivial constructs, mainly due to the global search they perform over candidate process models.

Taxonomies of Behavioral Process Models

In general, a taxonomy is a rooted tree where each node corresponds to a concept more general than those associated with all of its descendants. Taxonomical structures are widely recognized as a valuable technique for eliciting, consolidating and sharing relevant knowledge in disparate application contexts, which can profitably support a variety of tasks (see, e.g., References 1,8,9,29 for some works on this topic).

In this paper, we focus on the concept of *process taxonomy*, i.e., a tree of behavioral process models where the root provides the most abstract view over process executions, whereas any level of internal nodes encodes a refinement of this abstract model. In other words, each leaf models a different concrete behavioral class of process instances, whereas each non-leaf model (computed through some abstraction mechanisms) provides a unified representation for all the behavioral classes associated with its children. Notably, by enabling a semantic browsing of process models, such a structure can sensibly reduce the efforts for reusing and customizing a model and can make easier the exploitation of process knowledge.

Clearly, the semantic foundation of a process taxonomy is given by associating some suitable notion of specialization/generalization with parent-child relationships. In the remainder of this section, we first mention some major specialization/generalization notions introduced in diverse application fields for process models. Subsequently, we discuss an alternative generalization notion, which hinges on the definition of abstraction relationships between process activities, which will be considered in the remaining sections of the paper.

Process Specialization as Inheritance of Behavior

Diverse notions of specialization were defined in several application contexts, e.g., OO-Design/Programming[27,36,38], Enterprise Modelling[31], and Workflow Modelling[7,28].

The possibility of defining taxonomies for business processes was first considered in Reference 31, where a repository of process descriptions is envisaged to support both design and sharing of process models. In this kind of reasoning, viewing processes as class hierarchies and activities as properties associated with those classes, each subclass inherits the values of properties associated with its super-classes. By the way, the framework in Reference 31 allows for a non-monotonic inheritance, in the sense that a subclass may delete some inherited property, as a sort of exceptions w.r.t. its super-class. Clearly, such an approach founds on a “static" representation of the processes which completely disregard the evolution of process instances over time.

The question becomes particularly intriguing if one turns to look at the behavioral features expressed by a process model, representing a finite set of legal executions. Several frameworks for precisely defining a specialization/generalization of a process model, according to some suitable behavioral semantics, have been proposed for different modelling formalisms, such as, e.g., Object Behavior Diagrams[38], UML diagrams[36], process-algebra specifications and Petri-nets[7], DataFlow diagrams[28]. Many of them generally state that all the execution instances of a model must also be instances of any model generalizing it.

A different meaning of inheritance is adopted in Reference 5, where two basic notions are defined w.r.t. workflow models represented as WF-nets (a special class of Petri Nets). In particular, Reference 5 states that the external behaviors exhibited by a model and by any of its specializations must not be distinguished whenever: (a) only common activities are performed (protocol inheritance, a sort of “invocation consistency"[36]), or (b) one abstracts from activities which are not in the original model (projection inheritance, a sort of “observation consistency"[36]).

Process Generalization via Activity Abstraction

An alternative way of expressing the generalization relationship between two process models consists in regarding the more general model as the result of some abstraction transformation applied to the more specific one, which essentially amounts to replace a group of activities (or an entire subprocess) with a single higher-level activity. This issue links to the topic of process abstraction[6,15,20,30,32] where the aim is to make more compact the representation of a process by providing the user with a more abstract and readable view.

In order to make the discourse more concrete, we next present, as an example, the simple and yet general framework introduced in Reference 13 to represent both partonomical and taxonomical relationships over process activities, in the form of an *abstraction dictionary*.

An *abstraction dictionary* is a tuple $D=\langle A,IsA,partOf\rangle$, where A still denotes a set of activities, while *IsA* and *PartOf* are binary relations over A. Intuitively, given two activities a and b, $(b,a)\in IsA$ indicates that b is a specialization of a, whereas $(b,a)\in PartOf$ indicates that b is a component of a. These basic properties are extended next in a transitive fashion. Given two activities a and x, a *implies* x if there is a path from a to x in the graphs induced by *IsA* and *PartOf*. In such a case we also say that a is a *complex* activity; otherwise, a is a *basic* activity. In a sense, complex activities constitute high-level concepts defined by aggregating or generalizing basics activities actually occurring in real process executions. This notion is the basic block for building a taxonomical process models where the knowledge about process behavior is structured into different abstraction levels.

An intuitive notion of process model generalization can be finally stated as follows: Given two workflow models W_1 and W_2, we say that W_2 *specializes* W_1 (W_1 *generalizes* W_2) w.r.t. an abstraction dictionary D, if for each activity a_2 of W_2 (i) either a_2 appears in W_1 or there is an activity a_1 in W_1 s.t. a_1 *implies* a_2, and (ii) there is no activity b_1 in W_1 s.t. a_2 *implies* b_1. Finally, given an abstraction dictionary D and a tree of workflow models G, G is a *model taxonomy* w.r.t. D if Wp *generalizes* W, for any pair of models W and Wp s.t. W is a child of Wp in G.

The definition of generalization stated above allows for obtaining a taxonomy of process models, even though it does not found on precise notions of execution semantics and behavioral inheritance for these models. Anyway, as it is parametric w.r.t. the kind of activity abstraction relationships, it is general enough to accommodate quite sophisticated forms of workflow generalization, provided that abstract activities and their mapping to concrete ones are defined accordingly. For example, one could well think of defining partonomical relationships according to some suitable activity aggregation scheme, such as the *SPQR*-tree[37], which encodes a hierarchical decomposition of a process model into process "fragments" (i.e. components) corresponding to well-defined control flow structures. In fact, by explicitly taking into account of control-flow constraints in the definition of process fragments, such an approach would allow for order-preserving process abstraction, whenever all concrete activities of a fragment are replaced with a single (abstract) activity.

Mining Different Behavioral Classes via Log Clustering

Clustering techniques can help recognizing different behavioral classes of process instances automatically, by exploiting the information captured in log data. In this section we first overview a series of clustering approaches recently defined for the specific case of workflow traces. We then illustrate their usage within a recursive partitioning scheme, in order to induce a hierarchy of execution classes (and associated behavioral models), as a core structure for eventually deriving a taxonomy representing the behavior in the log at different abstraction levels.

Basic Trace Clustering Methods

The idea of clustering log traces has been reckoned in the community of Process Mining as an key technique for the analysis of complex real-life processes exhibiting unstructured and flexible behaviours, which can hardly represented in a compact and effective manner through a single workflow model. In such a case, indeed, trace clustering allows for identifying homogeneous sets of traces that can be adequately represented with an easily interpretable process model.

A prevalent approach to clustering traces consists in transforming traces into vectors where each dimension corresponds to an activity[14,16,17]. Different methods were proposed to project log traces into such a feature space, most of which focus on the frequency of activities in the log traces. Clearly such a *bag-of-activities* representation, suffers, as a major drawback, from the loss of temporal information, in that it does not takes account of the ordering of activities. One way for alleviating this problem is to regard each trace as a sequence of activities and to extract a number of k-grams (i.e. subsequences of length k) from it, as features for the clustering[16].

This vector space model was combined with new context-aware features[19], by expanding the core idea of considering activity subsequences conserved across multiple traces. Unlike the *k*-gram approach, subsequences of variable length are detected which frequently occur in the log, and are assumed to correspond to some hidden functionalities of the process. Using these conserved subsequences as features, the clustering is expected to put together traces that are similar from a functional viewpoint.

A special kind of sequential features, named *discriminant rules*, was introduced in Reference 14 in order to capture unexpected behavioral patterns, which are not modelled properly by a given workflow model, and can hence help refine it by way of more precise and specific models. Precisely, a *discriminant rule* is a rule of the from $[a_1 \ldots a_h]$ -/-> a s.t.: *(i)* $[a_1 \ldots a_h]$ and $[a_h a]$ are both "highly" frequent (i.e., the frequency is above a given threshold σ), and *(ii)* $[a_1 \ldots a_h a]$ is "lowly" frequent (its frequency is below another threshold γ). In fact, discriminant rules can be straightforwardly derived from frequent sequential patterns, which in their turn can be discovered efficiently via a level-wise search strategy[3,4,22]. As an instance, the discriminant rule [fil]-/->m for the example process *HandleOrder*, captures the fact that a fidelity discount is never applied when a (new) client is registered – this constraint is not captured the worfklow model in Fig. 1.

The vector space model was still exploited in Reference 16, where the idea was proposed of clustering traces by considering *profiles*, that is a set of related items describing the trace from some specific perspective (i.e., activities, transitions, data, performance, etc). Each item is associated with a measure assigning a numeric value to any trace. Therefore, by transforming each log trace into a vector containing all these measures, any distance-based clustering method can be exploited to partition the log.

A different approach to trace clustering dismisses the vector space model in favour of syntactic techniques which operates on the whole sequence "as-is" by way of string distance metrics. For instance, a context-aware approach based on the generic edit distance[33] was proposed in Reference 18. The edit distance between two sequences is defined as the cost of the optimal combination of edit operation (insertion, deletion or substitution) that allows to transform one sequence into another. The cost of edit operations can be adapted to the peculiarities (primarily, concurrence nature) of workflow processes. This is done in Reference 18 by introducing an algorithm that automatically derive edit operation costs from log traces.

Discovering a Hierarchy of Process Classes

Hierarchical clustering schemes (agglomerative or divisive) have been extensively used in several application contexts in order to construct taxonomies automatically[8,29]. Traditionally, these schemes rely on suitable distance measures and/or linkage strategies, and produce a tree-like partitioning structure ("dendrogram"), which can serve as a basis to derive a hierarchy of classes.

As an alternative way, we next describe a top-down clustering method[14] specifically proposed for the case of process logs, and sketched in the algorithm *HierarchyDiscovery* in Fig. 3. This algorithm decomposes hierarchically the process model into a number of sub-processes, by iteratively splitting a cluster whose associated model is expected to mix different usage scenarios. The result is a tree-like model whose nodes correspond to set of executions and where, for each node, its children correspond to the splits originated by that executions.

Initially a single workflow model W_0 is extracted that is a first attempt to model the whole log. Iteratively, one of the models not refined yet (i.e., corresponding to a leaf of the tree) is refined: the set of traces that are associated with it are split into clusters by using function *Partition-FB*, which could implement one of the different trace clustering approaches discussed in the previous subsection. A new workflow model is then mined for each of these clusters, by using some workflow discovery technique like the one discussed above. At the end of the process, a hierarchy of workflow model is obtained, where the leaf nodes constitute a disjunctive model capturing the execution logs in a more accurate than W_0.

Figure 3

Algorithm *HierarchyDiscovery*.

INPUT: a log *L*, 2 numbers *maxSplit* and *K*, *a threshold* γ

OUTPUT: a model hierarchy ***H***, and an associated disjunctive model ***DW***

Method: Perform the following steps:

A) Initialize the hierarchy with one workflow model for the whole log

   W_0 = ***mineWFSchema(L)*** // mine a model W_0 from all traces in *L*

   Cluster(W_0)=*L* // associate W_0 with the whole log *L*

   DW={W_0} // initialize ***DW*** with W_0

B) WHILE *size*(***DW***)≤ *maxSplit and soundness*(***DW***)< γ

   Extract the least sound model *W** from *DW*

   Partition-FB(***Traces***(**W***))

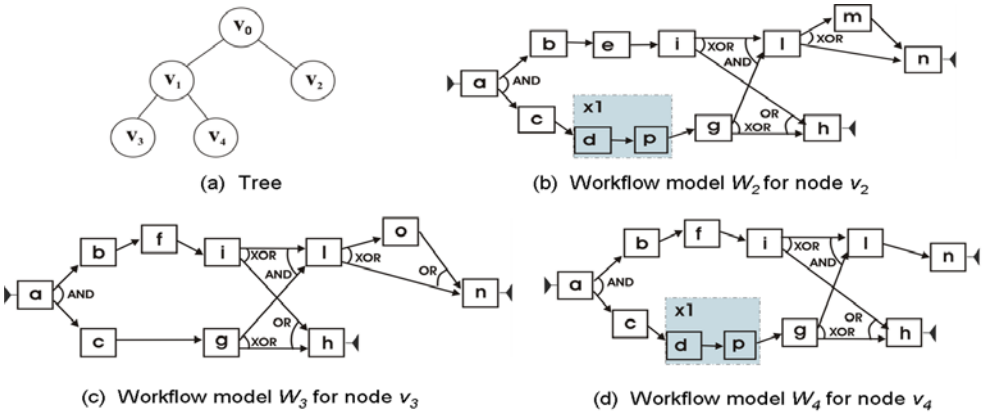
   For each cluster ***Cj*** (j=1..k) extracted from *Traces*(*W**),

   W_j= ***mineWFSchema(L)*** //mine a (refined) model for the cluster

   put W_j in ***DW***, and extend ***H*** by adding W_j as a child of ***W****

Figure 4

Hierarchy found by *HierarchyDiscovery* on the running example (details for leaf models only).



Example (contd.) In order to provide some insight on how the algorithm works, we report a few notes on its behavior over 100,000 traces randomly generated from the workflow in Fig. 1, under the additional constraint that task *m* cannot occur in any trace containing *f* (a fidelity discount in never applied to a new customer), and task *o* cannot appear in any trace containing *d* and *p* (fast dispatching cannot be performed whenever external supplies are asked for), simulating different usage scenarios that cannot be captured with a simple workflow model. The output of *HierarchyDiscovery*, for *maxSplit*=5 and γ=0.85, is the hierarchy shown in Fig. 4.(a), where each node logically corresponds to both a cluster of traces and a workflow model induced from that cluster. In particular, node v_0 corresponds to the whole log and to the preliminary workflow model induced from it. Since this model is not as sound as required by the user, the log is partitioned into two clusters (*k*=2). The cluster at v_2 is not further refined (due to its high soundness), whereas that at v_1 is split again. The models associated with the leaves of the tree are shown in Fig. 4.(b-c-d). In fact, models W_0 and W_1 (associated with v_0 and v_1, respectively) are only preliminary attempts to model the log traces, which are indeed modelled in a sounder way by the leaf models. Nevertheless, the whole hierarchy is an important result as well, and is a basis for deriving a model taxonomy, as discussed next.

Restructuring Workflow Hierarchies into Taxonomies

Given a hierarchy of workflow models (possibly extracted with algorithm *HierarchyDiscovery*), a crucial task is to restructure them into a taxonomy of models that describe the process instances at different levels of details. The key point is to equip each non-leaf node with an abstract model that generalizes those associated with all of its children. To this aim, some activity abstraction methods must be introduced to replace groups of activities with higher-level ones.

Figure 5

Algorithm *BuildTaxonomy*

```
INPUT:  a model hierarchy H
OUTPUT: the (modified) model hierarchy H, and an abstraction dictionary
          D, s.t. H is a model taxonomy w.r.t. D
Method: Perform the following steps
Initialize the dictionary D as empty
Create a set S of models containing only the leaves of H
WHILE there is a model v in H, s.t. v∉S and its children are in S
  v:=generalizeSchema(children(v),D) // replace v with a model
                                     // generalizing v's children
    put v in S
NormalizeDictionary(D,H)    // remove "superfluous" complex activities
```

A Bottom-Up Restructuring Scheme based on Activity Abstraction

An interesting approach to restructuring hierarchies into taxonomies was discussed in Reference 13. The crucial steps are illustrated in Fig. 5, via an algorithm, named *BuildTaxonomy*[13]. The algorithm takes in input a model hierarchy ***H*** and transforms it into a taxonomy, according to an abstraction dictionary ***D*** computed by the same algorithm. In a bottom-up fashion, the algorithm replace each non-leaf workflow model *v* in the hierarchy with a novel model that generalizes all the children of *v*. Such a generalization task is carried out by providing the procedure *generalizeModels* with the models associated with the children of v, along with the abstraction dictionary *D*, empty initially. As a result, a new generalized model is computed and assigned to *v*, while updating *D* in order to suitably relate the activities abstracted with the complex ones they were replaced with in the resulting generalized model. As a final step, the algorithm restructures the dictionary *D* by removing all "superfluous" activities that were created during the generalization. More specifically, any complex activity ***a*** appearing in no model of *H* is removed, as it can be abstracted into another, higher-level, complex activity *b*, provided that this latter actually implies all activities implied by ***a***.

The basic idea used in Reference 13 to generalize the models associated with the children of any node *v* consists in merging them all into an overall workflow model, and in replacing groups of "specific" activities (i.e., activities not appearing in all child models), with new "virtual" activities that represent them at a higher level of abstraction. In this way, only the features shared by all the children of *v* are represented exactly, while other activities are abstracted into new high-level (i.e., *complex*) activities. To this purpose, some suitable activity abstraction procedure must be defined that allows to aggregate groups of "specific" activities to be replaced with a (possibly new) complex activity abstracting them all via *IS-A* or *PART-OF* relationships.

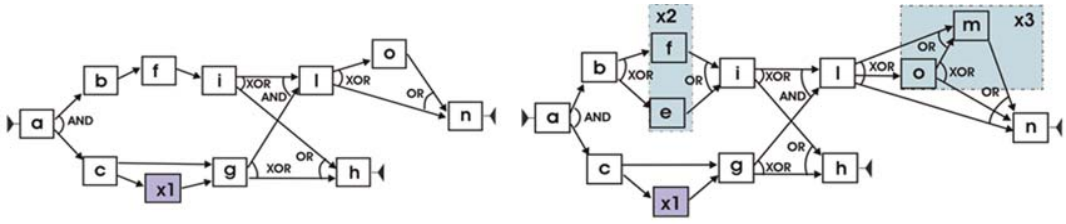
A heuristic method[13] for performing such abstraction iteratively selects a pair *(x,y)* of "specific" activities to be abstracted together into a single higher-level activity. Such a pair is chosen in a greedy fashion, by trying to minimize the number of spurious flow links that their merging introduces between the remaining activities and considering their mutual similarity w.r.t. the

contents of the abstraction dictionary *D*. This is done by resorting to a series of affinity measures assessing how much two any "specific" activities are suitable to be merged according the abstraction relationships already stored in *D: (i)* a "topological" affinity measure $sim^E(x,y)$, measuring how similar the neighbourhoods of *x* and *y* are w.r.t. the flow graph; and (ii) two "semantical" affinity measures, $sim_D^P(x,y)$ and $sim_D^G(x,y)$, expressing how similar x and y are w.r.t. the relationships of *IS-A* and *PART-OF*, respectively, stored in *D*.

All these measures are combined into an overall ranking function *score* defined as follows: $score(x,y)=0$, if (x,y) is not a merge-safe pair of activities; and $score(x,y)= max\ \{ sim^E(x,y), sim_D^P(x,y), sim_G^P(x,y) \}$, otherwise. Here a pair (x,y) of activities is said merge-safe (w.r.t. a given an set *E* of precedence relationships), if one of the following conditions holds: *(i)* there exist no path in *E* connecting *x* and *y; (ii) x* and *y* are directly linked by some edges in *E* and after removing these edges no other path exists between them.

Figure 6

Generalized workflow models in the taxonomy found for the example *HandleOrder* process.



(a) Workflow model W_1^* for node v_2 (b) Workflow model W_0^* for node v_2

Example (contd.) Consider again the hierarchy shown in Fig. 4. Algorithm *BuildTaxonomy* starts generalizing the leaf models W_3 and W_4 associated with v_3 and v_4, respectively. The result of this generalization is the model W_1^* shown in Fig. 6.(a), which is obtained by first merging all the activities and flow links contained in either W_3 or W_4, and by then performing a series of abstractions steps over all non-shared activities, namely *o*, *d* and *p*. When deriving the model W_1, only the activities *d* and *p* are abstracted, by aggregating them both into the new complex activity x_1; consequently, *d* and *p* are replaced with x_1, while the pairs (d, x_1) and (p, x_1) are inserted in the *PartOf* relationship. The model W_1 is then merged with the model W_2, and a new generalized model, shown in Fig. 6.(b) and named W_0^*, is derived for the root. In fact, when abstracting activities coming from W_2, *d* and *p* are aggregated again together, into a new complex activity x_2; however, in a subsequent step x_2 is incorporated into x_1, as these two complex activities have the same set of sub-activities and the same control flow links. Furthermore, the activities *e* and *f* are aggregated into the complex activity x_3, while *m* and *o* are aggregated into x_4. Consequently, the pairs (e, x_3), (f, x_3), (m, x_4) and (o, x_4) are added to the *PartOf* relation.

Further Abstraction Techniques

A large body of work was done concerning the transformation of workflow models via abstraction techniques, in order to simplify the representation of process behavior, by typically resorting to the aggregation of process activities[8,30,32]. In particular, in Reference 30, an abstract view of a workflow model is obtained automatically by replacing multiple real activities with "virtual" ones, based on ad-hoc aggregation rules, which guarantee that all original ordering relationships among the activities are preserved.

In Reference 32, an abstraction approach is proposed that relies on a partonomical decomposition, named *SPQR*-tree, of the given workflow model. In such a tree, each leaf node coincide with a single (atomic) process task, while any other node correspond to a "fragment" of the workflow model, featuring multiple process tasks related though different kinds of well specified composition pattern (which include sequence and split/join structures). The approach relies on a manual control by the user, who is in charge of specifying which

process task (or collection of tasks) in the original workflow model is to be abstracted. Based on a series of abstraction rules (specifically defined for each kind of composition pattern) the approach automatically replaces each task t indicated by the user with the finest grain workflow fragment encompassing t (i.e. the closest ancestor of t in the *SPQR*-tree). Clearly, the process can be iterated in order to produce higher level abstraction representation of the workflow.

Quite a different (log-driven) approach is adopted in some recent works[15,20] appeared in Process Mining community, where the main aim is to identify groups of behaviourally correlated activities directly from log traces, without considering any workflow model. The motivation is that often log systems track low-level events rather than semantically relevant process activities, and the raw application of process mining algorithms to such logs typically results in “spaghetti-like" models that are hard to comprehend. In particular, a sequence-based abstraction approach was proposed in Reference 15, where low-level events are grouped into clusters, which are regarded as an evidence for higher-level process activities. The core idea consists in identifying coherent subsequences of events within a trace and then using them to transform the trace into a sequence of event clusters. These clusters may be viewed as a categorization of events in types of events or event classes. Multiple abstraction levels are discovered for grouping events into classes by way of a hierarchical agglomerative clustering method, based on the proximity of event classes within log traces. Log traces are then transformed into sequence of abstract activities by choosing a cut of the resulting hierarchy of event clusters, and by replacing each event with its ancestor lying on that cut.

Another pattern-based activity abstraction approach[20] basically uses repetition patterns (e.g., tandem repeats) and sequence patterns from string-processing and bioinformatic literature, in order to capture loops and groups of correlated activities. The approach works in two phases: first, it extracts repetition patterns by looking at log traces individually, and then discovers common groups of activities by logically regarding the whole log as a sequence. More complex constructs such as choice and intra-loop parallelism are resolved by applying the pre-processing method on log traces iteratively. Efficient (suffix-tree based) structures are used to curb computation time. Moreover, to make the approach robust to the presence of both parallelism and choice constructs, a single abstract activity is created for patterns whose associated activity sets either contain each other or share many elements. Similarly to the previous approach, all of the discovered groups of activities are finally considered as high-level process activities, which can be used to transform the log traces, prior to the application of process mining algorithms.

Conclusion

We have discussed a series of basic modelling, mining and abstraction techniques that help organizing knowledge on a given process into a taxonomical structure, representing its behavior at various abstraction levels. In particular, some details have been presented for recent approaches that leverage on divisive clustering to automatically discover a hierarchy of behavioral classes. A taxonomy of models can be then derived by applying activity abstraction methods over all non-leaf nodes, which allow to eventually equip them with high-level models.

A number of challenging issues are still open and deserve being investigated. For example, the recognition of abstract activities can benefit from available background knowledge on the activities’ semantics, possibly extracted from a given thesaurus or a process ontology. Moreover, discovered process taxonomy can be exploited profitably to analyze relevant measures, such as usage statistics and performance metrics, along the different usage scenarios of the process at hand. Specifically, by using a taxonomy as an aggregation hierarchy for multi-dimensional OLAP analysis, it is possible to enable the user to interactively evaluate such measures over different groups of process instances. Notably, such an extension would be a valuable feature within interactive process mining settings, like the one considered in Reference Reference 21, which can effectively support the user in evaluating the discovered process models, as well as in tuning the parameters in subsequent mining

sessions. Finally, the discovered taxonomies can serve as a basis for further knowledge discovery tasks, such as the mining of generalized association rules between, e.g., the users or the resources involved in the workflow process under analysis.

References

1. Adami G, Avesani P, Sona D Clustering documents into a web directory for bootstrapping a supervised classification *Data & Knowledge Engineering* 2005, 54(3):301-325.

2. Agrawal, R, Gunopulos, D, Leymann, F. Mining process models from workflow logs. In *Proc. 6th Int. Conf. on Extending Database Technology (EDBT'98)*; 1998 469-483.

3. Agrawal, R, Imielinski, T, Swami, A. Mining association rules between sets of items in large databases. In *Proc. 1993 ACM SIGMOD Int. Conf. on Management of Data* (*SIGMOD'93);* 1993 207-216.

4. Agrawal, R, Srikant, R. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. on Very Large Databases (VLDB'94)*; 1994 487-499.

5. Basten T, van der Aalst WMP Inheritance of behavior *Journal of Logic and Algebraic Programming* 2001, 47(2):47-145.

6. Casati, F, Castellanos, M, Dayal, U, Shan, MC. iBOM: a platform for intelligent business operation management. In *Proc. 21st Int. Conf. on Data Engineering (ICDE'05)*; 2005 1084-1095.

7. Chen, CWK, Yun, DYY. Discovering process models from execution history by graph matching. In *Proc. 4th Int. Conf. on Intelligent Data Engineering and Automated Learning (IDEAL'03)*; 2003 887-892.

8. Chuang, SL, Chien, LF. A practical web-based approach to generating topic hierarchy for text segments. In *Proc. 13th ACM Int. Conf. on Information and Knowledge Management (CIKM'04)*; 2004 127-136.

9. Makris C, Panagis Y, Sakkopoulos E, Tsakalidis A Category ranking for personalized search *Data & Knowledge Engineering* 2007, 60(1):109-125.

10. de Medeiros, AKA, van Dongen, BF, van der Aalst, WMP, Weijters, AJMM. Process mining: Extending the *α*-algorithm to mine short loops. Technical Report *WP 113.* Eindhoven University of Technology; 2004.

11. de Medeiros AKA, Weijters AJMM, van der Aalst WMP Genetic Process Mining: An Experimental Evaluation *Data Mining and Knowledge Discovery* 2007, 14 (2):245-304.

12. Georgakopoulos D, Hornick M, Sheth A An overview of workflow management: from process modeling to workflow automation infrastructure *Distributed and Parallel Databases* 1995, 3(2):119-153.

13. Greco G, Guzzo A, Pontieri L Mining Taxonomies of Process Models *Data & Knowledge Engineering* 2008, 67(1):74-102.

14. Greco G, Guzzo A, Pontieri L, Saccà D Discovering expressive process models by clustering log traces *IEEE Trans. on Knowledge and Data Engineering* 2006, 18(8):1010-1027.

15. Günther, CW, Rozinat, A, van der Aalst, WPM. Activity Mining by Global Trace Segmentation. In *Proc. 7th Intl. Conf. on Business Process Management (BPM'09)*; 2009.

16. Song, M, Günther, CW, van der Aalst, WPM. Trace Clustering in Process Mining. In *Business Process Management Workshops (BPM'08)*; 2008 109-120.

17. de Medeiros, AKA et al. Process Mining Based on Clustering: A Quest for Precision. In *Business Process Management Workshops (BPM'07)*; 2007 17-29.

18. Jagadeesh Chandra Bose, RP, van der Aalst, WPM. Context Aware Trace Clustering: Towards Improving Process Mining Results. In *Proc. SIAM Int. Conf. on Data Mining (SDM'09)*; 2009 401-412.

19. Jagadeesh Chandra Bose, RP, van der Aalst, WPM. Trace Clustering based on Conserved Patterns Towards Achieving Better Process Models. In *Proc. 5th Intl. Workshop on Business Process Intelligence (BPI'09)*; 2009.

20. Jagadeesh Chandra Bose, RP, van der Aalst, WPM. Abstractions in Process Mining: A Taxonomy of Patterns. In *Proc. 7th Int. Conf. on Business Process Management (BPM'09)*; 2009.

21. Hammori M, Herbst J, Kleiner N Interactive workflow mining requirements, concepts and implementation *Data & Knowledge Engineering* 2006, 56(1):41-63.

22. Han, J, Pei, J, Yi, Y. Mining frequent patterns without candidate generation. In *Proc. Int. ACM Conf. on Management of Data (SIGMOD'00)*; 2000 1-12.

23. Herbst J, Karagiannis D Integrating machine learning and workflow management to support acquisition and adaptation of workflow models *Journal of Intelligent Systems in Accounting, Finance and Management* 2000, 9:67-92.

24. Herbst J, Karagiannis D Workflow mining with InWoLvE. *Computers in Industry. Special Issue: Process/Workflow Mining* 2003, 53(3):245-264.

25. IDS Scheer. ARIS Process Performance Manager (ARIS PPM): Measure, analyze and optimize your business process performance (whitepaper).

26. Junginger S, Kuhn H, Strobl R, Karagiannis D Ein geschafts-prozessmanagement-werkzeug der nachsten generation - ADONIS: Konzeption und anwendungen. *Wirtschaftsinformatik* 2000, 42(3):392-401.

27. Kueng, P, Kawalek, P, Bichler, P. How to compose an object-oriented business process model (http://www.cs.man.ac.uk/ipg).

28. Lee J, Wyner GM Defining specialization for dataflow diagrams *Information Systems* 2003, 28(6):651-671.

29. Li T, Zhu S Hierarchical document classification using automatically generated hierarchy *J. Intell. Inf. Syst.* 2007, 29(2):211-230.

30. Liu DR, Shen M Workflow modeling for virtual processes: an order-preserving process-view approach *Information Systems* 2003, 28:505-532.

31. Malone TW et al Tools for inventing organizations: Toward a handbook of organizational processes. *Management Science* 1999, 45(3):425-443, 1999.

32. Polyvyanyy, A, Smirnov, V, Weske, M. The Triconnected Abstraction of Process Models. In *7th Int. Conf. on Business Process Management (BPM'09)*; *2009* 229-244.

33. Ristad ES, Yianilos P.N Learning String-Edit Distance *IEEE Trans. Pattern Anal. Mach. Intell. 1998*, 20(5):522-532.

34. Schimm G. Mining most specific workflow models from event-based data. In *Proc. of Int. Conf. on Business Process Management (BPM'03)*; 2003 25-40.

35. Schuldt H, Alonso G, Beeri C, Schek H Atomicity and isolation for transactional processes *ACM Trans. Database Syst. 2002*, 27(1):63-116.

36. Stumptner, M, Schrefl, M. Behavior consistent inheritance in UML. In *Proc. 19th Int. Conf. on Conceptual Modeling (ER'00)*; 2000 527-542.

37. Weijters AJMM, van der Aalst WMP Rediscovering workflow models from event-based data using Little Thumb *Integrated Computer-Aided Engineering* 2003, 10(2):151-162.

38. Wen, L, Wang, J, Sun, JG. Detecting Implicit Dependencies Between Tasks from Event Logs. In Proc. of the 8th Asia-Pacific Web Conference (APWeb'06); 2006 591-603.

39. van der Aalst, WMP. The application of Petri nets to workflow management. *Journal of Circuits, Systems, and Computers* 1998, 8(1):21-66.

40. van der Aalst, WMP, Desel, J, Kindler E. On the semantics of EPCs: A vicious circle. In *Proc. EPK 2002: Business Process Management using EPCs*; 2002 71-80.

41. van der Aalst WMP et al Workflow mining: A survey of issues and approaches *Data & Knowledge Engineering* 2003, 47(2):237-267.

42. van der Aalst WMP, Weijters AJMM, Maruster L Workflow mining: Discovering process models from event logs *IEEE Transactions on Knowledge and Data Engineering* 2004, 16(9): 1128-1142.